MIT 9.520/6.860, Fall 2018
Statistical Learning Theory and Applications

Class 10: Neural Networks (aka Deep Learning)

Lorenzo Rosasco

Learning functions

So far:

- Linear

$$f(x) = w^\top x.$$

- Features

$$f(x) = w^\top \Phi(x) = \sum_{j=1}^{p} w_j \varphi_j(x).$$

- Kernels

$$f(x) = \sum_{i=1}^{n} K(x, x_i) c_i.$$

Learning functions (cont.)

Random features: $\Phi(x) = \sigma(Sx) = (\sigma(s_1^\top x), \ldots, \sigma(s_M^\top x))$

$$f(x) = w^\top \sigma(Sx) = \sum_{j=1}^{M} w_j \sigma(s_j^\top x),$$

where:

- $\sigma : \mathbb{R} \to \mathbb{R}$ can be nonlinear.
- $(s_1, \ldots, s_M)^\top = S$ random.

Neural networks (shallow)

$$f(x) = w^\top \sigma(Sx) = \sum_{j=1}^{M} w_j \sigma(s_j^\top x),$$

where:

- $\sigma : \mathbb{R} \to \mathbb{R}$ can be nonlinear.
- $s_1, \dots, s_M = S$ free parameters.
- Offsets are typically added $\sigma(s_j^\top x + b_j)$.

Other neural networks

Other form of neural networks can be considered.

For example radial basis functions (RBF) networks

$$f(x) = \sum_{j=1}^{M} w_j \sigma(\|s_j - x\|),$$

e.g. $\sigma(\|s_j - x\|) = e^{-\|s_j - x\|^2 \gamma}$.

Neural nets vs features/kernels

$$f(x) = \sum_{j=1}^{M} w_j \sigma(s_j^\top x), \qquad f(x) = \sum_{j=1}^{M} w_j \sigma(\|s_j - x\|).$$

- Random features: any non linearity, random S.
- Kernels: pos. def. non linearity, S is the training set.
- Neural nets: any non linearity, any S. Extensions by composition.

Multilayer neural networks (deep)

$$f(x) = w^\top \sigma(S_L \ldots \sigma(S_2 \sigma(S_1 x)))$$

where

- $\sigma : \mathbb{R} \to \mathbb{R}$ can be nonlinear.
- $M_0 = d$ and $M_j \in \mathbb{N}$.
- $S_j = \mathbb{R}^{M_j} \to \mathbb{R}^{M_{j-1}}, \qquad j = 1, \ldots, L.$
- Offsets are typically added.

Building functions by composition.

Neural networks terminology

$$f(x) = w^\top \sigma(S_L \dots \sigma(S_2 \sigma(S_1 x)))$$

- Each intermediate representation corresponds to a (hidden) layer.
- The dimensionalities $(M_\ell)_\ell$ are the number of hidden units.
- The non linearity is called activation function.

Activation functions

- Logistic function $\sigma(\alpha) = (1 + e^{-\alpha})^{-1}$, $\alpha \in \mathbb{R}$,
- Hyperbolic tangent $\sigma(\alpha) = (e^{\alpha} - e^{-\alpha})/(e^{\alpha} + e^{-\alpha})$, $\alpha \in \mathbb{R}$,
- Rectified linear unit (ReLU) $\sigma(\alpha) = |s|_+$, $\alpha \in \mathbb{R}$.

Neural networks function spaces

The space of functions of the form

$$f_{(w,(S_\ell)_\ell)}(x) = w^\top \sigma(S_L \ldots \sigma(S_2 \sigma(S_1 x)))$$

does not have a linear structure, hence no inner product/norm.

Compare to features/kernels.

What now?

- Why neural networks?
- Computations.
- Extensions.

Why neural networks?

- Learning representations.
- Biological interpretation.
- Because they are awesome.

Today we discuss the first two items.

Learning data representations

$$f_w(x) = \Phi(x)^\top w$$

So far Φ fixed a priori.

Can it be learned?

Learning linear representations

We need to parametrize Φ.

Consider Φ linear, that is
$$\Phi(x) = Sx.$$

Then, consider
$$f_{(w,S)}(x) = w^\top Sx,$$
where both w and S need to be estimated.

Learning linear representations (cont.)

$$f_{(w,S)}(x) = w^\top Sx,$$

Different constraints can be imposed on w, S.

Still, we are working with linear functions,

$$\beta = S^\top w \quad \mapsto \quad f_{(w,S)}(x) = \beta^\top x.$$

Beyond linear representations

$$f_{(w,S)}(x) = w^\top S x,$$

- Insert non linearity in front of x

$$f_{(w,S)}(x) = w^\top S \Phi(x), \quad \mapsto \text{back to kernels...}$$

- Insert non linearity in front of S

$$f_{(w,S)}(x) = w^\top S \Phi(x), \quad \mapsto \text{neural nets}$$

Biological interpretation

One neuron

$$z = \sigma(s^\top x) = \sigma(\sum_{j=1}^{d} s^j x^j)$$

- Each neuron has d inputs.
- Each input is multiplied by a different weight *stored* in the neuron.
- The weighted inputs are aggregated by summation.
- The activation function suppresses small outputs and clips big outputs.

Connecting neurons

One neuron

$$z = \sigma(s^\top x) = \sigma(\sum_{j=1}^{d} s^j x^j)$$

Another neuron taking as input other neurons,

$$y = \sigma(w^\top z) = \sigma(\sum_{j=1}^{d} w^j z^j)$$

Connectionism is deep

One neuron

$$z = \sigma(s^\top x) = \sigma(\sum_{j=1}^{d} s^j x^j)$$

Another neuron taking as input other neurons,

$$y = \sigma(w^\top z) = \sigma(\sum_{j=1}^{M} w^j z^j)$$

Another neuron taking as input other neurons,

$$u = \sigma(v^\top y) = \sigma(\sum_{j=1}^{N} v^j y^j)$$

...

Computations

$$f_{(w,(S_\ell)_\ell)}(x) = w^\top \sigma(S_L \dots \sigma(S_2 \sigma(S_1 x)))$$

ERM for neural nets

$$\min_{w,(S_\ell)_\ell} \sum_{i=1}^{n} (y_i - f_{(w,(S_\ell)_\ell)}(x_i))^2,$$

possibly with norm constraints on the weights (regularization).

Computations

$$f_{(w,(S_\ell)_\ell)}(x) = w^\top \sigma(S_L \dots \sigma(S_2 \sigma(S_1 x)))$$

ERM for neural nets

$$\min_{w,(S_\ell)_\ell} \sum_{i=1}^{n} (y_i - f_{(w,(S_\ell)_\ell)}(x_i))^2,$$

possibly with norm constraints on the weights (regularization).

The problem is non-convex and possibly non smooth depending on σ.

Optimization for neural nets

Let

$$\widehat{L}(w,(S_\ell)_\ell)=\sum_{i=1}^n (y_i - f_{(w,(S_\ell)_\ell}(x_i))^2.$$

Gradient descent becomes

$$\begin{aligned}
w^{t+1} &= w^t - \gamma_t \partial_w \widehat{L}(w^t,(S^t_\ell)_\ell)\\
S_L^{t+1} &= S_L^t - \gamma_t \partial_{S_L} \widehat{L}(w^t,(S^t_\ell)_\ell)\\
\dots &\\
S_1^{t+1} &= S_1^t - \gamma_t \partial_{S_1} \widehat{L}(w^t,(S^t_\ell)_\ell)
\end{aligned}$$

The step-size $(\gamma_t)_t$ are often called learning rates.

There is a natural order to compute derivatives by the chain rule.

Computations for shallow network

Consider a one hidden layer network and corresponding ERM,

$$\widehat{L}(w, S) = \sum_{i=1}^{n} (y_i - f_{(w,S}(x_i))^2,$$

Unrolling all the equations in gradient descent we get

$$\begin{aligned} w_j^{t+1} &= w_j^t - \gamma_t \partial \widehat{L}_{w_j}(w^t, S^t) \\ \cdots & \\ S_{j,k}^{t+1} &= S_{j,k}^t - \gamma_t \partial \widehat{L}_{S_{j,k}}(w^t, S^t) \end{aligned}$$

Back-propagation & chain rule

By direct computations,

$$
\begin{aligned}
\partial \widehat{L}_{w_j}(w,S) = &= -2\sum_{i=1}^{n} \underbrace{(y_i - f_{(w,S)}(x_i)))}_{\Delta_{j,i}} \sigma(s_j^\top x_i) \\
&\ldots \\
\partial \widehat{L}_{S_{j,k}}(w,S) = &= -2\sum_{i=1}^{n} \underbrace{(y_i - f_{(w,S)}(x_i)))w_j \sigma'(s_j^\top x)}_{\eta_{i,k}} x_i^k
\end{aligned}
$$

Back-prop equations: $\eta_{i,k} = \Delta_{j,i} w_j \sigma'(s_j^\top x)$

Using the above equations, iterations are performed in two steps:

- Forward pass: compute function values keeping weights fixed,
- Backward pass: compute errors and propagate
- Hence the weights are updated.

Few remarks

- Multiple layers are treated analogously: efficient derivative computations are needed.

- Stochastic gradient descent is the method of choice.

- A number of variations are considered, including
  - acceleration,
  - minibatching,
  - batch normalization
  - ....

Auto-encoders

- A neural network with one input layer, one output layer and one (or more) hidden layers connecting them.
- The output layer has **equally** many nodes as the input layer,
- It is trained to **predict the input** rather than some target output.

Auto-encoders (cont.)

An auto encoder with one hidden layer of k units, can be seen as a **representation-reconstruction** pair:

$$\Phi : X \to \mathbb{R}^M, \quad \Phi(x) = \sigma\left(Sx\right), \quad \forall x \in X$$

with $M < d$ and

$$\Psi : \mathbb{R}^M \to X, \quad \Psi(\beta) = \sigma\left(\tilde{S}\beta\right), \quad \forall \beta \in \mathbb{R}^M.$$

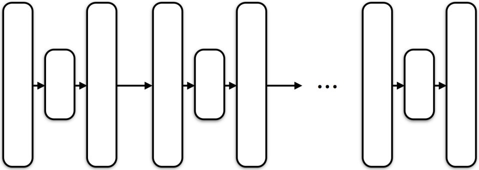
ERM corresponds to find best data reconstruction

$$\min_{S,\tilde{S}} \sum_{i=1}^n \left\| x_i - \Psi \circ \Phi(x_i) \right\|^2$$

Stacked auto-encoders

Multiple layers of auto-encoders can be **stacked** [Hinton et al '06]...

$$\underbrace{(\Phi_1 \circ \Psi_1)}_{\text{Autoencoder}} \circ (\Phi_2 \circ \Psi_2) \cdots \circ (\Phi_\ell \circ \Psi_\ell)$$



...with the potential of obtaining **richer** representations.

Pre-training

- Unsupervised training of each layer to initialize supervised training.

- Potential benefit of unlabeled data.

Convolutional neural networks

From

$$f(x) = \sum_{j=1}^{M} w_j \sigma(s_j^\top x),$$

to

$$f(x) = \sum_{j=1}^{M} w_j \left\| \sigma(s_j \star x) \right\|_\infty ,$$

where:

- pooling

$$\|\beta\|_\infty = \max_j |\beta_j|,$$

- convolution

$$(s \star x)_i = \sum_{j=1}^{d} s_j x_{i-j}.$$

Convolutions

Convolution is linear

$$s^\top x = C_s x$$

C_s is a circulant matrix (each row is a shifted copy of s).

Weights sharing

Let

$$C_S = (C_{s_1}, \dots, C_{s_M}).$$

Then

$$(\sigma(s_1 \star x), \dots, \sigma(s_M \star x)) = \sigma(C_S x)$$

a standard neural nets with repeated weights.

Pooling

Compared to classical neural nets CNN have structured nonlinearities.

Other form of pooling can be considered

- average pooling

$$\frac{1}{d}\sum_{j=1}^{d}\beta_j,$$

- ℓ_p pooling

$$\|\beta\|_p = \left(\sum_{j=1}^{d}|\beta_j|^p\right)^{\frac{1}{p}}.$$

Why CNN?

- Invariance.
- Hierarchical compositionality.